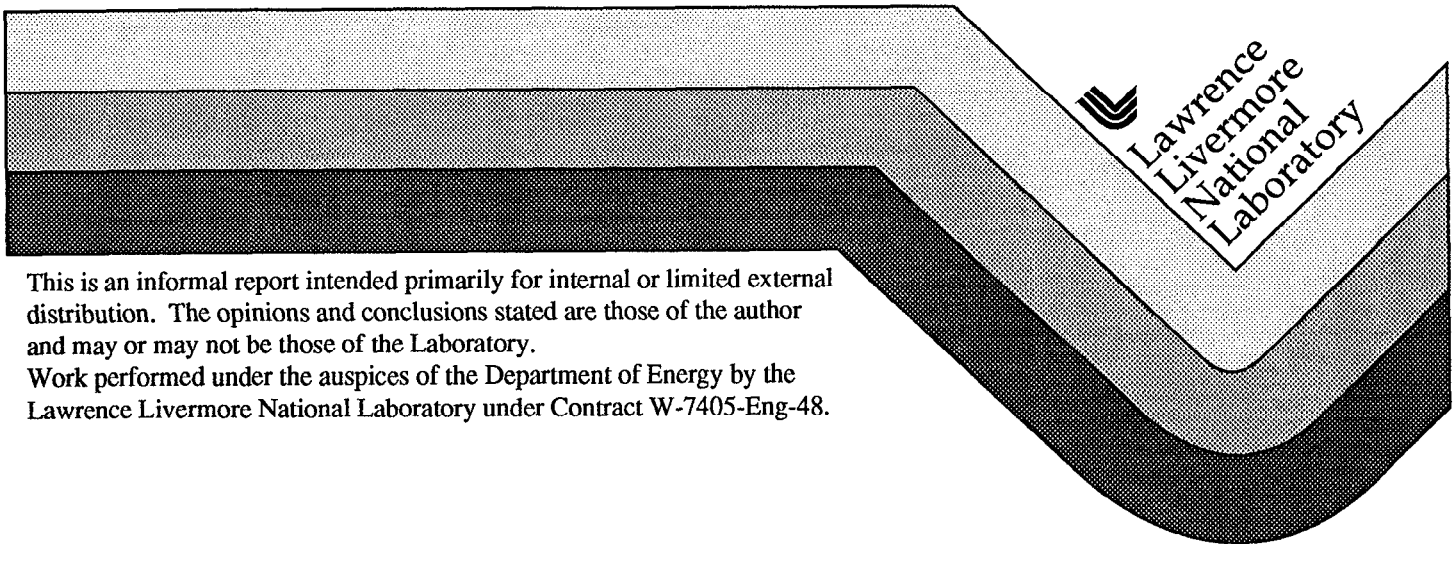
UCRL-ID-129903

Supporting Large-Scale Computational Science

Ron Musick

October 1, 1998



Work performed under the auspices of the Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Supporting Large-Scale Computational Science*

http://www.llnl.gov/*/dbms/index.html

Ron Musick
Center for Applied Scientific Computing
rmusick@llnl.gov
10/98

Abstract

A study has been carried out to determine the feasibility of using commercial database management systems (**DBMSs**) to support large-scale computational science. Conventional wisdom in the past has been that DBMSs are too slow for such data. Several events over the past few years have muddied the clarity of this mindset:

1. Several commercial DBMS systems have demonstrated storage and ad-hoc quer access to Terabyte data sets.
2. Several large-scale science teams, such as EOSDIS [NAS91], high energy physics [MM97] and human genome [Kin93] have adopted (or make frequent use of) commercial DBMS systems as the central part of their data management scheme.
3. Several major DBMS vendors have introduced their first object-relational products (**ORDBMSs**), which have the potential to support large, array-oriented data.
4. In some cases, performance is a moot issue. This is true in particular if the performance of legacy applications is not reduced while new, albeit slow, capabilities are added to the system.

The basic assessment is still that DBMSs do not scale to large computational data. However, many of the reasons have changed, and there is an expiration date attached to that prognosis. This document expands on this conclusion, identifies the advantages and disadvantages of various commercial approaches, and describes the studies carried out in exploring this area. The document is meant to be brief, technical and informative, rather than a motivational pitch. The conclusions within are very likely to become outdated within the next 5-7 years, as market forces will have a significant impact on the state of the art in scientific data management over the next decade.

* This work was performed under the auspices of the U.S. Department of Energy at LLNL under contract no. W-7405-Eng-48.

Table of Contents

1 Introduction

Business needs have driven the development of commercial database systems since their inception. As a result, there has been a strong focus on supporting many users, minimizing the potential corruption or loss of data, and maximizing performance metrics like transactions per second, or TPC-C and TPC-D results. These optimizations have little to do with the needs of the scientific community, which typically revolve around a great deal of compute and I/O intensive analysis, often over large data with high dimensionality. In addition, these optimizations also have little to do with supporting various business intelligence needs such as the decision support and data mining activities common in on-line analytic processing (**OLAP**) applications. As a result, business data have typically been off-loaded to secondary systems, then processed with specific analytic and data mining tools. For scientific data, in many cases the data was never collected in a DBMS in the first place, and so the analysis and visualization takes place on the original flat-file format. This is a painful solution, because a DBMS has much to offer in the overall process of managing and exploring data.

Of late, industry and the research community have been pushing to develop DBMS-based systems that will break this mold, and provide the needed support OLAP. The recent activity in OLAP [GC97], multi-dimensional databases [TD96], ORDBMS [SM96], and the TPC council's TPC-D [TPC98] benchmark all testify to the strength of this new direction. This is a promising change of focus. OLAP optimizations are much closer than on-line transaction processing (**OLT**) to supporting the interactive computational data analysis (**ICDA**) activities that take place in scientific domains. OLAP and ICDA do not, however, represent the same type of workload. In fact, little is known about these differences, and about exactly how DBMS technology fails to meet ICDA needs. We explore this issue in some depth, describing an evaluation of DBMS technology for large high-dimensional computational data from ASCI. We have found that the technology is much closer to being able to support this activity than many think. Furthermore, there is a fairly clear evolution path that should lead to full support once the technology catches up to ICDA requirements. This might happen with the major vendors within 5-7 years.

This report serves several purposes. The first is to introduce the particular stresses that computational science place on a data management infrastructure. This highlights how and where available DBMS technology does not fit ICDA requirements. Second, the data being generated in ASCI has many commonalties with computational data from other scientific domains, and to a large extent, to OLAP data. Understanding it will help make sense of ICDA application needs, and may also provide insight into future requirements of business-oriented OLAP data. Third, there are no stable and well-known benchmarks that serve the area of ICDA. We do not attempt to create one. However, the ideas, descriptions and issues that are described provide useful input that could lead to such a benchmark in the future.

The paper is written for an audience with a background in computer science, although much of the document is accessible to anyone with a technical background. Throughout this document there are pointers to code, schemata, case studies, or other details of this study that can be found on the web. The rest of the paper is organized as follows. Section 2 describes some related work. Section 3 details the characteristics of computational data, the corresponding implications for ICDA, and the potential benefits of using commercial technology. Section 4 is the heart of the document. It summarizes the issues and advantages of the various DBMS options in a list of brief statements. Then in the second half of Section 4, the supporting data for each of those statements is explained in detail. Section 5 closes with a discussion of new technologies that are worth keeping track of, research and development directions, and pointers to further information.

2 Related Work

For the past several years, DBMS performance measurements have been dominated b the tests supported by the Transaction Processing Council. The two current benchmarks supported by the Council are TPC-C and TPC-D (A and B are already defunct). TPC-C [TPC98] measures the OLTP performance of a system, and so provides information on transactions-per-second (TPS) throughput rates for simple queries in multi-user settings. TPC-D measures OLAP performance, providing TPS and cost/performance information within the context of multiple users running complex queries that include updates to the original data. There are other benchmarks that measure different aspects of DBMS performance for different types of applications, such as those found in Gray's handbook [Gra93], and those offered by the OLAP council [OLA98]. The value of any benchmark, though, is strongly dependent on how closely it matches the workload of typical applications of interest to an organization. As described in detail in Section 3, the match between OLAP, OLTP and ICDA is not strong.

The bulk of performance papers published in recent years have been geared towards measuring specific tasks or sub-systems within a DBMS. Some examples of this include work on topics like pointer swizzling [WD92], or extensible arrays [RZ96]. There have also been a multitude of in-house studies in industry that focus on measuring performance for specific applications. The results of these studies add to the lore that surrounds the database world, but they are very difficult make practical use of. In general, there is little information to be found concerning broadly applicable performance studies for large-scale scientific applications.

There are several descriptions of the data from other scientific domains, along with the corresponding difficulties in managing that data. In particular, earth observing satellite data (EOSDIS) has been described in many places [BS95, NAS91]. This data is similar to the ASCI data described below, but the desired exploration mechanism differs, as well as the source of the data, and the (very important!) cultural aspects that control how the data is accessed and utilized. The human genome project, which involves small, highl complex data with different analysis requirements, has been described in [Goo95]. The data collected in the high energy physics community is different in form from

computational data, and has access patterns that are similar in many respects to OLAP [MM97]. Several other large-scale applications and the respective OODB solutions are described in [CL97]. This paper concentrates on how to support ICDA tasks on data generated by computational scientists.

3 Computational Data

ASCI data is primarily mesh data, which is one of the most basic and commonly used conceptual models for describing physical systems with computer models. A mesh provides a way of breaking a surface or volume down into an interconnected grid of much smaller zones (see Figure 1). Each zone stores a range of computed or collected variables. The hope is that if the zones are small, the micro-scale properties and interactions can be modeled with enough accuracy to provide precise predictions of macro-scale events. The hope of improving the accuracy of the overall model by creating meshes with larger numbers of smaller zones leads to irresistible urges to create meshes with extremely fine granularity. The main limitation is the additional storage and computational costs incurred by increasing the number of zones in a mesh. Current hero-level capabilities are at the scale of a few billion zones; a more typical range is between tens of thousands, to tens of millions of zones. In general, the computational data we generate will always be at the edge of our computing and storage capability.

Characteristics

The mesh data is typically stored in highly structured binary flat-files, using standard low-level self-describing formats like NetCDF [RDE93] or HDF [HDF98]. The mesh is accessed through high-level API's (such as Silo or Exodus) that provide methods to read and write individual components of the mesh directly, without reading the rest of the structure into memory. The data itself is also highly structured. The 2 and 3-dimensional zones in a mesh are made up of lists of points, or nodes. For example, a cubic zone has 8 nodes that describe the corners of the cube. Nodes have X, Y, and coordinates, whose values can change if the particular mesh deforms over time. Variables can be assigned to zone centers, or to nodes. Variable values are recorded at each time step in a simulation, for each node and zone in the mesh. These are the basic components of any mesh, and there are several ways to extend this general framework. For example, for efficient visualization, lists are stored that record the zone faces which are on the surface of a mesh (and therefore need to be drawn), as well as edgelists that help when performing certain visualization computations.

A few typical visualization-oriented operations are: (1) pseudo-color plots of a variable colored according to its value at different nodes and zones; (2) select and view an iso-surface in the mesh; (3) view an orthogonal slice or vector plot; and (4) play a movie of different time steps of the mesh. Each new operation is hand-coded and manuall inserted into the visualization tool before being able to be used by a scientist. This domain, in Stonebraker's terms [SM96], is a quadrant 3 and quadrant 4 problem. There is complex data, with the occasional need to support ad-hoc queries.

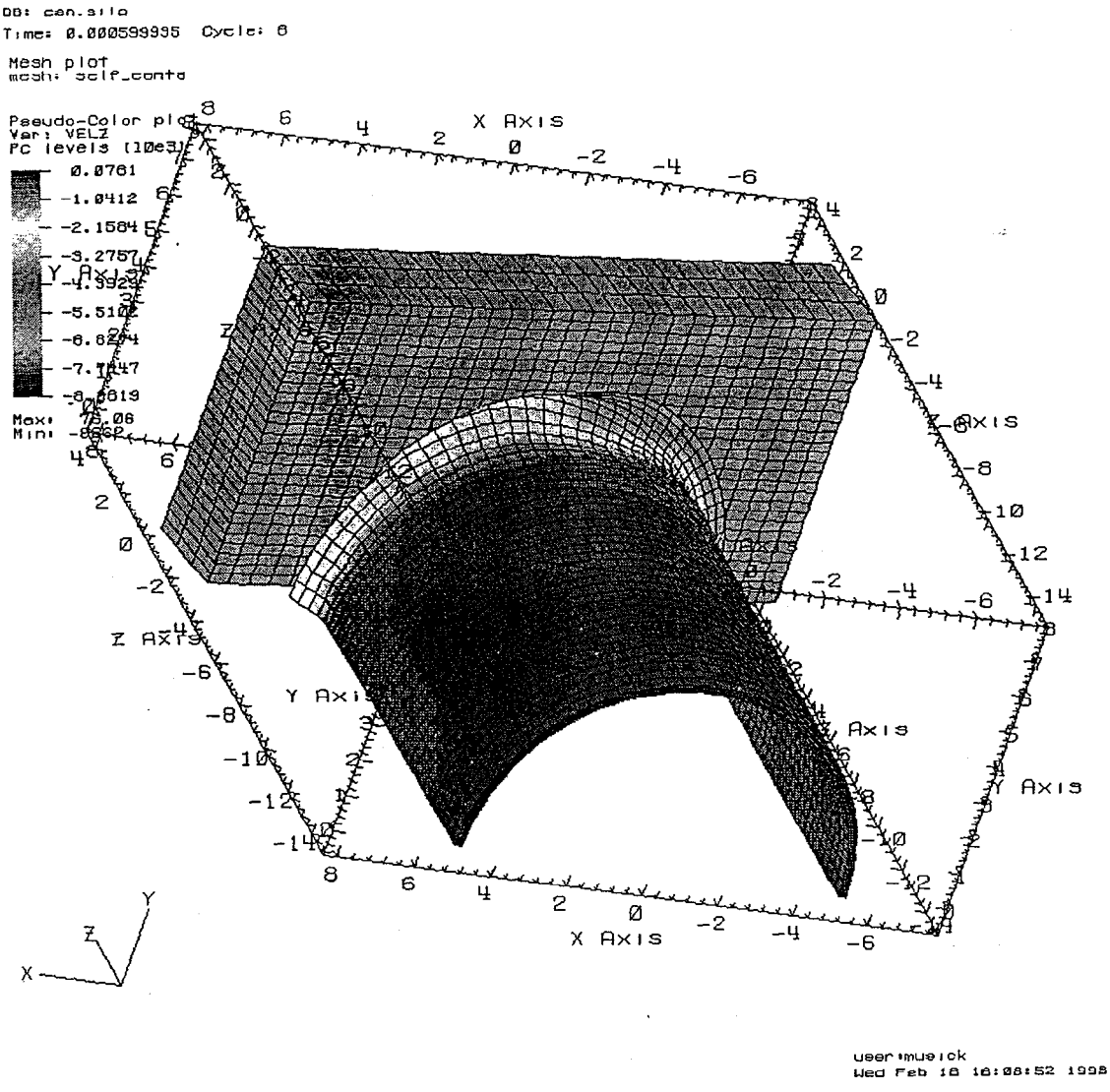


Figure 1

This is a rendering of a small mesh depicting a can being crushed up against a wall. In this figure, there are more than 10 variables stored in every zone; the velocity in the Z direction is being displayed in this pseudo-color plot. This snapshot is taken from the 8th time step of a movie sequence that ends up with the can completely crushed against the wall. This is a pseudo-color plot of the velocity in the Z direction.

ICDA data management implications

The computational data described above is representative of typical ICDA (interactive computational data analysis) applications. The characteristics of a ICDA are clearl distinct from OLAP and OLTP. To summarize:

- *Few concurrent, local users.* OLTP-style optimizations geared towards improving the overall throughput for multiple users may negatively impact single user performance. ICDA users are local, and there is little or no concurrent use of the same file. The expected system loads depend on the purpose of the machine in question, but even for the general servers, there will most likely be fewer than ten people accessing, browsing, or computing with the mesh data at any one time.
- *Write Once, Read Many.* Any DBMS optimizations geared towards making updates (insert, delete) faster may impede WORM performance.
- *Transactions are not needed.* The top 2 conditions largely obviate the need for transactions. Optimizations that are aimed at improving TPS ratings are not meaningful here. In fact, it would be preferable to turn transactions off if that part of the code path in a DBMS could be avoided.
- *High dimensional, dense data.* High dimensional business data is typically sparse, which can lead to some optimizations built into multi-dimensional database solutions that do not generalize to this data.
- *Emphasis on pure data throughput - Mb/s.* Visualization of large data requires throughput rates comparable to native Unix. The tasks that the new data management system would need to support include the visualization capabilities that already exist, and the query-driven exploration capabilities that the quer interface would enable. The performance requirements of the two tasks are not the same, as depicted in Figure 2. Supporting visualization involves supplying large amounts of data, quickly, to code that draws a mesh, or computes iso-surfaces of, say, a zone variable. The primary queries involved in this type of work are likely to be range queries, where high throughput (Mb/s) is a must. Supporting the ad-hoc queries, on the other hand, is more likely to involve point, multipoint, extremal and other classes of SQL queries. While high data throughput is helpful here, a good query optimizer plays a more central role.
- *Large data.* The data will be as large as the computational and storage platforms allow.

There are extensive DBMS evaluation matrices available [CL97] to help make the choice of which DBMS to use, once it has been decided that a DBMS is an appropriate data management system for a task. These allow comparisons of platform support, transaction semantics, transactions per second, blob support, java support, and etc. Before this analysis is useful however, we must first answer whether a DBMS is a good choice. Understanding the characteristics of ICDA applications is a good first step to being able to answer that question, which we have explored above. Understanding the culture, and

the expectations of the ICDA community is the next piece of information needed to address the utility of the DBMS approach here.

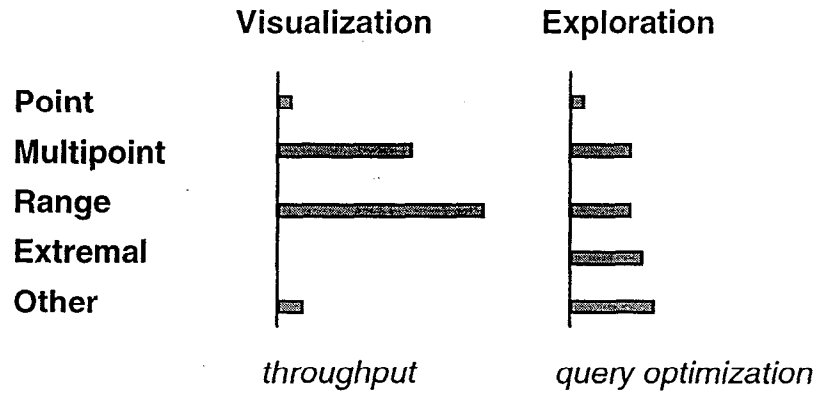


Figure 2

Estimated query spread for high-throughput visualization queries, and for highly complex lower-throughput query-driven exploration of the dataset. Particular examples of queries can be found in the next section.

The ICDA culture has a significant impact on the required data management and computational infrastructure. One aspect is that independent of how large or fast computers and storage devices get, the available system will always be used to its limits by choosing meshes with ever-increasing numbers of zones. ASCI machines have already reached more than 1 sustained Teraflop, and this year should see more than 3 TF. Storage will be in place by the end of the year that includes at least 25 TB of disk, about 1 TB of main memory, and over a petabyte of tertiary storage. The ASCI machines will, however, be quickly used at capacity. The typical pattern of usage, common to man large supercomputing centers, is to select a job, and give it full access to a significant portion of the available disk and compute resources. The job will run, using the bulk of these allotted resources. Once finished, the mesh and resulting data will be off-loaded to tertiary storage to make room for the next run. Several boundary conditions on a data management solution fall out of understanding this mode of operation.

1. *Excellent performance on legacy applications*
   First, the underlying data management system must be fast. The typical ICDA system is based on API's built over specialized binary data formats (such as Silo and Exodus). These systems are fast. For example, Silo runs within 50%, and usuall within 90% of the peak throughput (in MB/s) of Unix fread/fwrite performance for sequential reads/writes of very large files using optimal request sizes. For ASCI, an new data management system would need to be within at least 50% of Silo or Exodus performance, or the users will not accept it no matter what the new capabilities are.

This requirement must be treated carefully, however. There are ways to use new ORDBMS technology in a manner that does not impact legacy performance at all. Such an approach is described in the case study below. If the performance of legacy applications is untouched, then the user might be willing to put up with poor performance when 1) the capability being provided is new, and 2) there is no other more practical or efficient way to get the answer to that query.

2. *Efficient use of storage*
   Second, due to the large physical size of the generated mesh data, the approach needs to be efficient in its use of storage space, and needs to be interfaced to a tertiary store. In particular, doubling or tripling the size of the original binary file is simply not an option. If space was not such an issue, a copy of the binary file could be loaded into a DBMS. Existing visualization functionality could be provided from the original data with no reduction in performance. New query-driven exploration functionality could be supplied from the data in the DBMS. Given the stringent storage requirements, however, even the cost of creating multiple indexes (let alone multiple copies) would have to be watched closely.

3. *Security*
   Security can be a significant obstacle to any information system that brings groups of people together. The DBMS vendors have proven their systems capable of managing confidential business and financial data. They need to be re-validated on every closed system which has unique restrictions on which low-level communication protocols are allowed and which are not.

4. *Stability of the approach*
   Finally, business issues play an important role. The more isolation that can be achieved from a single vendor solution, and from the consequences of a vendor failure, the better. This means that a DBMS would need to have access to (and comply with) widely accepted standards for communication and querying, and that the product would need to be offered by a strong vendor.

Potential benefits of DBMS technology

There are several powerful draws to commercial technology that make it worthwhile to find a path to merge the requirements of ICDA and business computing.

1. *Advanced data exploration capability, such as ad-hoc querying through SQL*
   The value of our data rests on our ability to explore and analyze it to achieve scientific goals. The most immediate draw of DBMS technology is the enhanced data exploration possible with the integration of an ad-hoc query facility (accessed through SQL) and a visualization tool like that used to create Figure 1. For example, the following types of queries would be very useful to an ASCI scientist, but they are not currently supported in ASCI:

   Example 1: Highlight all zones and their nearest neighbors for which zone variable *pressure* $> \alpha$ but acceleration in the X axis is *xaccel* $< \beta$

Example 2: Display the bounding box that includes these zones, or cut the rest of the mesh out and only display the selected zones.

Example 3: Return a list of meshes for which, from time step 1 - 5, zone variable *pressure* was greater than α for at least 3 time steps.

Example 4: Of the meshes returned, what were the initial of each of the material mixes

This type of exploration would be a valuable extension of the capabilities available to the domain scientists who are responsible for understanding and characterizing large datasets. In order to provide that capability without DBMSs, one would need to develop an in-house query interface built on top of the existing data models and formats. The optimizer would involve a significant amount of effort.

2. *Ease of generating new queries*
   Flexibility is a key benefit of DBMS technology. The ad-hoc query mechanism gives the user a simple interpreted language with which to interact with the data. To ask a new query, the user writes a new SQL query. The query on the DBMS side can then be shared and used on any other dataset with the same relational model. Current approaches to ICDA tools are not very flexible; each new type of query basicall requires some off-line development before the scientist can run it. The query on the ICDA side can not be shared because the underlying data model and formats are likel to be different.

3. *Being in a position to profit from the forces driving the computer industry*
   The database market is an 8 billion dollar market this year, and growing rapidly. The Palo Alto Research Foundation has the data warehousing and data mining markets at 15 billion and growing at upwards of 40% a year. The major computer vendors have made it clear at high performance computing forums such as the Salishan conference that this business market is what is and will be driving their efforts, rather than the much smaller scientific computing community. If the ICDA community could make use of these commercial systems for data management and access, then the forces driving the computer industry would be driving the industry in directions more directly relevant to the ICDA community. This impacts high performance computers, available tools, and the availability of relevant technical skills in the workforce,.

4. *Access to a wealth of tools developed by 3rd party vendors, and academic research*
   There are a host of exploration, analysis, documentation, integration and organizational tools created by 3rd party vendors that work on the major commercial DBMS systems. Being able to buy and use such tools off the shelf is much more cost-effective than building them in-house for each different ICDA data model and format in a scientific community. There is also a very large and active community of academic database researchers; access to this large research community has several intangible benefits ranging from workforce skills to mainstream research with immediate impact on ICDA problems.

5. *Portability of applications through industry-wide standards*

Current ICDA visualization and other exploration tools are tied to the underlying data formats. These formats are not standard, and the result is that much work is duplicated across the National Labs and the scientific community as a whole in writing code for similar functionality over similar data stored in different ways. SQ (Standard Query Language) is an industry standard language for accessing business data stored in relational DBMSs. The RDBMS data model and query language is based on relational algebra, and relational calculus, both of which have strong mathematical underpinnings. There are also wide-spread communication protocols across DBMS systems such as ODBC and JDBC. Access through standard interfaces to data at multiple levels would allow the ICDA community to readily share data, tools, and labor.

6. *Distributed, parallel data manipulation for ``free"*
   The major DBMS vendors provide parallel DBMSs where data can be partitioned b rows or columns or tables across multiple disks in a system. Tasks can also be distributed to multiple server instances. The query optimization routines then (attempt to) determine optimal access patterns to that data. Furthermore, as computing clusters and business needs evolve, market forces will require DBMS capabilities to evolve as well in order to stay competitive. Parallel capabilities in current ICDA environments require a great deal of in-house effort to achieve an acceptable level of performance and accuracy.

7. *Access to a diverse collection of current and future capabilities*
   There are a large number of other capabilities that are linked to commercial DBMS technology. Some may have value for scientific data management, such as: persistence, CORBA, COM, direct integration of the web with the DBMS, digital library interfaces, standard metadata repositories, extensible systems that can handle multimedia and quality of service, spatial data blades or cartridges, data cleaning and transformation tools, and etc. The enormous size of the DBMS market, and a fruitful research community provide a large range of options and extensions to the basic storage management and access capabilities of a DBMS.

4 Commercial options and viability

This section contains a summary of conclusions, followed by supporting arguments for some of the statements made. The statements preceded by "**" are described in depth in the second half of the section. Several prototype implementations, small-scale performance studies, and design experiments have been carried out in generating these conclusions. It seems that in this industry, the ratio of promotional hype to reality is so high that without such experimentation, it is very difficult to get good information at anything but a gross level.

Summary of Options

The pros and cons of using the three major categories of commercial DBMS approaches for ICDA applications are considered here. The most prevalent systems by far are

relational. RDBMSs have been on the market since the early 70's. Object-oriented and object-relational DBMSs are considered in here as well; these make up a much smaller market segment. The object-relational approach is likely to outgrow relational and object-oriented approaches, and become predominant over the next decade.

- **ODBMS**

Advantages

1. The object model is a powerful approach to data modeling, making it easy to build models that are representative of the data, and more naturally fit the structure of data in the scientific domain.
2. Navigational queries are much faster than in relational systems, and association queries (find data based on its characteristics) can be faster. This is due to less of a focus on OLTP workloads and ad-hoc query processing, and a data model more suitable for array processing.
3. These systems point the way for RDBMS, and to an extent ORDBMS evolution.

Disadvantages

4. There is a very weak standards base for the object query language (OQL), and the object definition language (ODL) [Cat96]. Each vendor has a different quer interface to the data, which tends to tie applications to a single vendor. This results in less stable products, and limits the number of 3rd party tools available.
5. The market for OO databases has always been small. With the major RDBMS vendors making very strong ORDBMS bids, a serious danger is that the ODBMS market will dry up.
6. Security can be a big issue. The ODBMSs more focused on providing fast, persistent object stores for C++ may provide high performance by having the storage system working within the same address space as the program (thus avoiding a context switch for simple updates). Malicious user programs can be written that will use this to access system calls.

Myth

7. **"ODBMSs are RDBMSs with object capabilities". Actually, man ODBMSs are primarily persistent object storage systems for C++, smalltalk, JAVA and other OO programming languages.
8. **"ODBMSs are fast". The speed advantage can be hard to realize. The speed depends greatly on the schema, the query structure, the physical layout or clustering of the data on the disk, the optimization routines, the underlying DBMS architecture, and so on.

ODBMS Conclusions: ODBMS has had several technical advantages in the past, but has not been able to capitalize on them. Now much of the technical advantage has been lost with the onset of object-relational systems from the major vendors. As time goes on, the ODBMS market seems poised to wither away. Of the potential benefits for ICDA moving to commercial technology listed in Section 3, choosing ODBMS is not likely to result in #3, 4, 5 or 7 coming true. Moving to ODBMS at this point in time does not seem beneficial.

- **RDBMS**

Advantages

1. The RDBMS market is very strong and growing, and the major vendors are relatively stable.
2. Standard interfaces to RDBMS data have been in place for a long time, at several levels of abstraction. This is what has enabled #4 below.
3. RDBMS products are relatively mature, which has many advantages including a large pool of experienced labor, knowledgeable customer support from vendors, fairly efficient software and well understood query optimization technology.
4. There is a large base of 3rd party tool vendors that offer a wide range of software for accessing and manipulating data stored in RDBMS. This wide base of tools makes an RDBMS the basis for a highly flexible, end-to-end data management capability, from design to storage and finally access.

Disadvantages

5. **Performance of these systems is very poor for ICDA, independent of the vendor. In the performance studies run below, slowdowns of 5 to 40x were seen for simple range and multipoint queries critical for ICDA.
6. **A significant drawback of relational systems is the clumsiness of the data modeling paradigm. For example, representing, accessing and computing with a multidimensional array is an extremely painful exercise in a relational system. The model is capable of representing arbitrarily complex objects, but the result is often far from intuitive, or efficient. These complexities can be hidden with views, for an additional layer of computing.

Myth

7. "Stable code" Any complex piece of software that is optimized to perform well on specific platforms will never be stable as long as the underlying platforms are replaced every other year or so.
8. **"Scientific data is too complex for RDBMSs". Existence proofs exist in the human genome domain, and EOSDIS that contradict this claim. Complexity of the underlying data is certainly an important issue, but alone is not sufficient to disregard this approach for data management needs.
9. **"RDBMSs are fast due to excellent query optimization benefits" This is just simply not true, as the performance study below details. Query optimization speeds things up, certainly, but does not get close to the capabilities of the I/O subsystem. DBMSs are still overwhelmingly compute-bound.

RDBMS Conclusions: The strengths of RDBMSs cover the weaknesses of ODBMSs. The main difficulty, however, is in the area of performance. RDBMS solutions require too much space, and are too slow to support legacy ICDA applications. This problem is independent of which vendor is used; it falls out as a result of the relational model, and the requirements of their OLTP customers. These approaches are not practical for ICDA currently, and they are not likely to be in the foreseeable future.

- **ORDBMS**

Advantages

1. **ORDBMS combines, in theory, the modeling capabilities of ODBMSs with all the RDBMS advantages mentioned above.
2. **ORDBMSs have fantastic potential for speed. As tested below, the approach is still only within 3-15x of native Unix. However, the approach is capable of providing near-native Unix fread and fwrite performance levels.
3. ORDBMSs have been in the planning stages for the major DBMS vendors (Oracle, Informix, IBM) for several years now, and the advantages of merging object and relational approaches have been proclaimed by the research communit for much longer. There is little doubt that ORDBMSs are "the wave of the future" as Stonebraker so aptly argues [SM96]. Nearly all of the potential for DBMS-oriented growth lies in this area, at levels that bode well for market share, vendor support, and 3rd party tools.
4. **This approach will be part of what eventually unites the ICDA and the OLAP data management systems.

Disadvantages

5. **There are no tested, bullet-proof approaches yet for how to integrate OO data into the relational query interface. This means that any application that relies on one style of interface will be largely tied to a single vendor.
6. **This is a new approach, and the instantiations of it being offered by the major vendors are not ready for general consumption. There is no delivery yet on the potential advantages mentioned above.
7. Vendor support is limited since none of the support staff understands the new technology in any depth.
8. **The interface to the new capabilities is buggy, fragile, limited, and at times completely bewildering.
9. Performance is hard to achieve, and can not yet deliver on the potential the approach shows.

Myth

10. **"Saves time" At this point, utilizing the OR aspect of the system to provide access to ICDA data is so difficult, that the time might be better spent being amortized over many smaller projects focused at extending legacy ICDA systems to allow ad-hoc queries, indexes, and etc.
11. **"More flexible" In building the OO data into the relational system, several restrictions are placed on the object – views if you like. The views limit the flexibility that might otherwise exist in the object. Furthermore, the access method that is created to support the external object will only be effective under certain rigid conditions.
12. "OO-level modeling" Full object-oriented models are not yet supported. Furthermore, in building the external data into the relational interface, much of the object-oriented aspect of that data is lost. SQL, the relational DBMS interface, is

still the primary interface to the data, with the result being a somewhat less satisfying middle-ground than one could hope for.

ORDBMS Conclusions: The potential for this technology is striking in terms of performance, flexibility, and ties to the business marketplace. This technology is onl a few years old, and as such there are several issues to be worked out, as described above. Currently, it is not ready for prime time ICDA applications. This is not due to limitations of the approach or different goals of the main customer base (as with RDBMS), or due to market, standards or query support problems (as ORDBMS). The difficulties stem primarily from the novelty of the approach. The main danger for ORDBMSs is failure of the approach due to the complexity of making use of it.

This option should be revisited in 5-7 years. By then, chances are that the technolog will be fully capable of providing all the benefits listed in Section 3. In fact, this technology could be the basis for a more radical merging of widespread scientific data, as underlying data models that support ICDA data such as the Vector Bundle Interface could be built and utilized efficiently from within an ORDBMS system.

Supporting Arguments

This section provides more in-depth arguments that support the statements made above that may be more contentious, or were reached as result of a more in-depth case study. The case studies are described below, and more information can be found on the web at: http://www.llnl.gov/*/dbms/index.html. This web site contains code samples, performance tests, schemata, talks, and so on.

ODBMS 7 Myth:

"ODBMSs are RDBMSs with object capabilities". Actually, many ODBMSs are primarily persistent object storage systems for C++, smalltalk, JAVA and other OO programming languages.

C++ is a weak OO language with no persistence. Man ODEMSs provide a full OO wrapper and persistence to C++ in a manner that hides the large bulk of the interaction with the DBMS from the user. Unlike relational systems, ODBMS do not focus on OLTP, and thus typically can not support large numbers of concurrent users, or high transaction loads. Also, due to the lateness of the ODMG group coming out with standards [Cat96] for object querying and the object definition language, it is only ver recently that the OO vendors have begun to show standard OQL or SQL interfaces to their data. The very lack of a standard query interface in part shows the much smaller focus ODMBS vendors have on supporting ad-hoc queries, as compared to their focus on providing a persistent OO layer for programming languages.

ODBMS 8 Myth:

"ODBMSs are fast". The speed advantage can be hard to realize. The speed depends greatly on the schema, the query structure, the physical layout or clustering of the data on the disk, the optimization routines, the underlying DBMS architecture, and so on.

Navigational queries in a ODBMS are fast, as they are pointer traversals in physical address space, compared to the hash table and index lookups that need to be done in a relational system. Association queries, however, can be slow. Associations involve finding data by the properties listed in an SQL"where" clause. In these cases, the advantages over RDBMSs are not striking. Performance depends on how the data is physically clustered on disk. ODBMSs typically allow clustering by objects, but ad-hoc queries over small portions of many objects (such as what would be seen in the high energy physics community) are troublesome, especially for those vendors that provide no support for building indexes. The schema definition can lead to difficulties, especiall where foreign key references are used, rather than object ids. As with any DBMS, the performance tuning knobs are so many that it takes an expert (or a lot of time) to squeeze acceptable performance out of the system.

---

*ODBMS * Conclusion:*

> *ODBMS has had several technical advantages in the past, but has not been able to capitalize on them. Now much of the technical advantage has been lost with the onset of ORDBMSs from the major vendors. As time goes on, the ODBMS market seems poised to wither away. Of the potential benefits for ICDA moving to commercial technology listed in Section 3, choosing ODBMS is not likely to result in #1, 3, 4, or 7 coming true. Moving to ODBMS at this point in time does not seem beneficial.*

It is difficult to make a strong argument for ODBMSs. The navigational performance of ODBMSs is excellent, and given good clustering, associational performance can be relatively good as well. But, the ODMG has only recently agreed on a set of standards for the community [Cat96], and many of these are still not fully implemented by the major vendors. Schema evolution (a very important task in evolving scientific domains [CGM98]) is much more difficult within ODBMSs. There is very limited 3rd part vendor support, and in general, the very future of ODBMSs is very unclear with the entrance of the major relational vendors into the ORDBMS market.

RDBMS 5 Disadvantage:

> *Performance of these systems is very poor for ICDA, independent of the vendor. In the performance studies run below, slowdowns of 5x to 40x were seen for simple range and multipoint queries that are common for ICDA.*

Performance Testing Considerations

The standard that a new DBMS-based implementation must compare favorably to is the existing legacy system in use on the scientist's desktop. For ICDA applications, this typically means comparing to native unix fwrite and fread performance with optimal block sizes (**NU**). For the purposes of this study, single node performance only is compared; it is a good indicator of what parallel results would be, since both NU and DBMS can be parallelized in similar ways. It might be reasonable for a DBMS-based system to expect that coming within 30%-50% of NU performance would be acceptable on the low end for some users to take an interest in the new capabilities being offered.

Generating fair and repeatable comparisons between any two systems is hard due to the number of variables that impact the validity of a comparison. Good benchmarks would help, but as mentioned in Section 2, there are no widely accepted benchmarks available for testing ICDA types of applications on DBMSs. This section does not define a new benchmark for ICDA. The goal has been instead to develop a set of tests that help clarif the performance implications of supporting ICDA-style queries with a NU-based approach, compared to asking the same set of queries against a DBMS-based approach. For the evaluations carried out in this part of the study, NU and DBMS performance is measured against the different classes of SQL queries laid out in Section 3. For each query class in Section 3, several simple, generic queries were constructed, examples of which can be found at http://www.llnl.gov/*/dbms/file/measurements.2.

Several issues come up when comparing two such open-ended approaches to data management. Some are well known, for example the tests should be run on the same system, and under the same conditions with respect to system load. The test suite should reflect a workload similar to the type of application that the system is meant to support. Some of the more interesting issues include:

- *Counting:* Throughput in terms of Mb/s is a key measure of interest for this evaluation. Exactly what data is counted when coming up with that throughput measure? For example, if the system reads 1Mb of useful information from a 3Mb interleaved array (but had to read the entire 3Mb to get the 1Mb), was the total amount of data read 1Mb or 3Mb? In the testing performed for this work, the answer is based on the *apparent throughput to the application*, and therefore is 1Mb.
- *DBMS Implementation Details:* DBMS tuning plays an important in determining performance. First, for each type of query, several different schemata were tested, with several ESQL implementations of the query, and using several different combinations of indexes. This resulted in several hundred small-scale performance tests. The DBMS tests were run using ESQL through a C interface, since that is the likely interface for scientific applications. Tuning was performed on the system according to principles and hints found in [Sha92], and vendor-specific sources. The goal of the tuning was not to squeeze the last 20% out of the system, but to be careful enough in the tuning, configuration and testing so that full range of performance was understood. Examples of the actual DBMS (and NU) testing code can be found at http://www.llnl.gov/*/dbms/rdbms_perf.html. The architecture describing how a DBMS would fit into an ICDA system is shown at http://www.llnl.gov/*/dbms/arch0.jpg.
- *NU Implementation Details:* For NU testing, the data was arranged on disk in binary flat-files in much the same way it would be arranged for Silo or Exodus files. For a given query, the NU implementations were written to retrieve the desired data as effectively as possible. Several different NU implementations were written per SQL query to elucidate reasonable bounds on different potential NU performance. The implementation of the NU tests raises a few interesting

issues. First, upper bounds on disk I/O speeds can be established with freeware programs available on the Web that do excellent jobs of measuring overall disk performance. For example, a program called ``Bonnie'' will measure character write, block write, rewrite, character read, and block read speeds. These numbers are not, however, suitable to be used as the throughput measurement for NU implementations. This is because of the wa *counting* is done; interleaving, or reading large amounts of metadata or useless data will result in *apparent throughput to the application* that is much lower than that predicted by Bonnie.

Second, choosing a reasonable NU implementation is a difficult task. One can always construct a NU test that will outperform the corresponding DBMS test on any query. An index can be constructed in C for NU, just as it can be in SQL for DBMS. But, whereas constructing and using an index in SQL is easy, doing the same from scratch in C takes much more energy, and so is usually done only in situations where there is a clear need. Because of this, three classes of NU tests were written for every SQL query: **poor**, **good**, and **very good**. As an example, consider a point query on a large array. The **poor** implementation basically reads through the array stored in binary on the file system, checking the condition of the query against each row of the array in turn. Each check is a system read. This linear search is the simplest, but also the slowest approach to this problem. The **good** implementation reads large chunks of the binary file into memory, checks for the proper condition, and continues until the proper datum is found. Finally, the **very good** implementation builds and uses an index. For an application in the current ASCI environment, if a particular point query is going to be used ver often, then the **very good** approach could be built into the visualization code. Without a typical DBMS-style query interface, however, it would be much more expensive to support fast, ad-hoc point queries to large multidimensional data.

- *Cache.* It is important to be able to separate out the impact of caching and memory on the performance of the storage hierarchy. The difference between a hot and cold cache can change the throughput by an order of magnitude or more. Isolating the cache allows one to predict what will happen with I/O performance as applications move from small-scale to large-scale, or toward tasks with low locality of reference (like some visualization tasks). To measure this impact, some operating systems provide specific commands that disable or clear the cache system. For others, mmap can be used to clear out the cache independent of memory size and cache associativity.

RDBMS Results

The tests performed over the course of the evaluation clearly demonstrated that RDBMSs do not provide the throughput performance needed for ICDA applications. The results summarized below are meant to give a flavor for the types of results that were generated in this process. These results are not meant to provide a comprehensive picture of the performance capabilities of any particular DBMS system; only a well written (and broadly accepted) benchmark for ICDA could do that.

The data in Figures 3 and 4 are from point and range queries, respectively. Point queries are the types of queries that one would expect a DBMS to consistently perform well on, mainly due to query optimization, and the relative ease of constructing and using indexes. The tests were constructed on fairly simple point queries with 2-4 conditions in the **where** clause. Indexes were constructed based on those conditions. The size of the data being stored ranged from several hundred kilobytes, to several hundred megabytes. Multiple operating systems, and multiple DBMS vendors were tested during this evaluation. Finally, the tests were conducted with both hot and cold caches.

As shown in Figure 3, for the point queries, the DBMS performed basicall on-par with the **good** NU implementations. When looking at NU throughput numbers, none of the implementations achieved more than 7% of the character read I/O capacity for the disk, as measured by Bonnie. The bulk of the tests showed results that utilized only 1-3% of the expected I/O bandwidth. This is not surprising, given that only apparent throughput to the application is being counted.

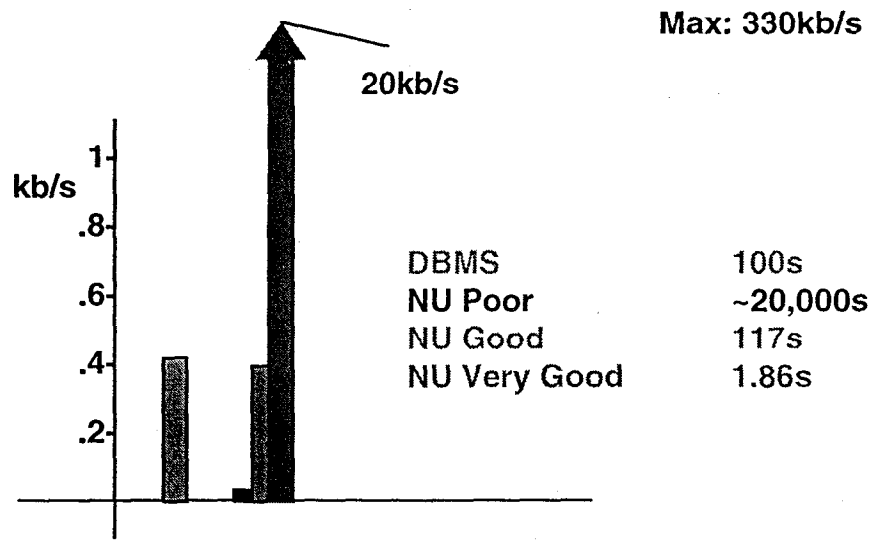


Figure 3

This figure depicts the throughput (in kb/s) achieved for point queries on a production server with several Mb of data. The NU tests were run on the same machine. The maximum character-per-second throughput rate (330kb/s) measured by Bonnie is low here. The numbers changed on faster machines, but the qualitative comparison between NU and DBMS did not.

The range and multipoint queries (Figure 4) were much less consistent between vendors than the point queries, and when compared to **good** NU implementations, total throughput rates varied from 5 to 40 times slower, even with substantial tuning. These tests were carried out with several different DBMS products, and on several machines

with data sizes that ranged up to 150Mb of "useful" data out of 500Mb total. The NU implementations reached maximum potential throughput within the first several Mb of data being read. The DBMS implementations flattened out much sooner, and at much lower throughputs.

It was interesting that the throughput rates achieved by different vendors varied on the order of 50-100%[1]. It turns out that the main difference comes from how often the DBMS server is contacted. For example, for one vendor, a C structure can be passed into the ESQL statement, and directly loaded with one call to the server. A second major vendor has no other mechanism but to fill up 32kb buffers one tuple at a time, and send that buffer via a cursor. This defeats the potential speedups that could be gained from range queries that scan clustered data on the disk. Another interesting note is that for small data, whereas the NU implementations ran anywhere from 10-50 times faster with a warm cache, the DBMS implementations consistently gained only about 10%. For larger datasets, the cache had much smaller beneficial effects, in particular for the NU tests. This most likely reflects the fact that the RDBMS is a compute-bound system.

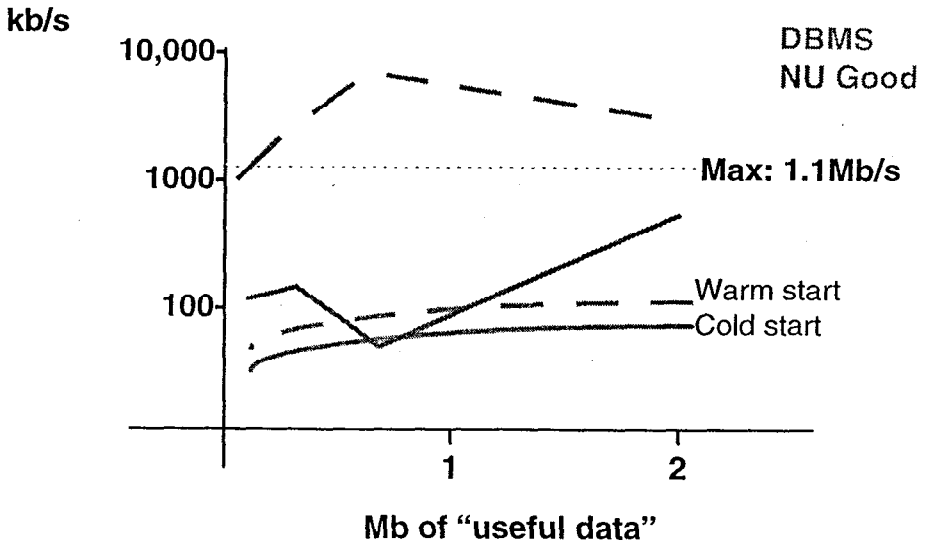


Figure 4

This figure depicts throughput results for range queries that had to retrieve several Mb of "useful" data. This particular test was run with all the data being useful to the query. The throughput for this system was also fairly poor, with block reads reaching only 1.1Mb/s max. This chart shows 5-8x performance dropoff on cold starts. For larger datasets on faster machines we saw 10x to 40x performance dropoffs, independent of vendor. For more information, see http://ww.llnl.gov/*/dbms/file/dmf97.ppt.

[1] While large, this variance does not substantially impact the overall conclusion of these tests (RDBMS performs poorly for ICDA data).

RDBMS 6 Disadvantage:

> *A significant drawback of relational systems is the clumsiness of the data modeling paradigm. For example, representing, accessing and computing with a multidimensional array is an extremely painful exercise in a relational system. The model is capable of representing arbitrarily complex objects, but the result is often far from intuitive, or efficient. These complexities can be hidden with views, for an additional layer of computing.*

The basic type system in an RDBMS is very simple and straightforward. The relational model represents data with a collection of *tables,* each of which is composed of a *set* of unordered *tuples.* Each table has a set of columns or *features* that define the table, for which every tuple has a value. Each feature can contain a basic data type such as char, int, string, date, float, and etc. Each table has defined for it a *primary key,* which is the subset of features that uniquely defines a tuple in a table. For a more in-depth introduction, see [ZMF97]. Some major elements of this model are: 1) it is set-based data, rather than ordered data; 2) there is no implicit memory indexing (see below) as found in most programming languages; 3) there is no mechanism for specifying methods, inheritance, or other object-oriented modeling features; and 4) there is a only a limited notion of abstract types. The extent of conceptual mismatch between this data model and a standard one that would be found in ICDA application code means that in using a relational system, the user will be required to maintain *two* data models – that of the application, and that of the DBMS. The translation code between the two can be non-trivial, and can easily reach thousands of lines of code.

In the relational model, even the simple definition of a vector becomes tedius. Since each table is officially a *set* of tuples (rather than an ordered list), the only way to insure that the jth element of array X is returned is to have a table that stores an index as the primar key, along with a second feature that is actually the value of X. This increases the required storage space, and increases the complexity of accessing elements of X in a query. As a brief example in the 1 dimensional case, to represent X(j), a table could be constructed with the name Table_X, a primary key (or feature) "index", and the value: *Table_X(index, value .* To find the value of X(3), one would need an SQL query similar to:

```
Select value from Table_X where index = 3;
```

While this is not too bad, it grows much worse in multi-dimensional cases. On the positive side, once in a table, we can ask very interesting, ad-hoc questions over the values and the indexes for X. If the return set is small (like in a point query, or a small multipoint or range query) and if the indexes have been built correctly, the returned answers can be on par with, or faster than a good NU implementation.

Another uncomfortable aspect is the typing that must occur in the schema. In the table Table_X above, the feature "value" must be declared as a float, an int, or etc. at schema definition time. A mesh, though, may have zone and node variables of mixed types. Representing this flexibility in a relational schema is messy, or requires a different

schema per mesh. Furthermore, many of the optimizations enabled by the RDBMS quer optimizer will not work unless the types of all the arguments match.

Most RDBMS vendors extend the basic relational system to some extent. Some extensions are introduced below. For the most part, while they are welcome additions to an otherwise spartan data model, they do not go far enough to comfortably support complex structured data in a natural way. For example, blobs (binary large objects) of some form or another are common. Blobs are large, uninterpreted bundles of bits, in some cases up to multi-Tb sizes. They are good for storing images, or in some cases large arrays. They can be attached to tables as a feature in many cases. The main drawback is that while access to the contents can be based on offsets from the start of the file, there is no SQL access to the interior of a blob, and thus blobs are useless when queries over that data are desired. Other extensions include: date and money native types (for business data); stored procedures that represent compiled (i.e. optimized) SQ queries; triggers for consistency checking; and complex types that allow the definition of a row type as a vector of simple types for use in schema definition.

There are formal methods for constructing normalized BCNF schemata. We did this for mesh data as handled in the Silo format, then extended it for reasons of efficiency and flexibility. The schema, and code to load this schema from a silo file can be found at `http://www.llnl.gov/*/dbms/index.html`. It might illuminate some of the issues that are mentioned in this section.

RDBMS 8 Myths:

> *"Scientific data is too complex for RDBMSs". Existence proofs exist in the human genome domain, and EOSDIS that contradict this claim. Complexity of the underlying data is certainly an important issue, but alone is not sufficient to disregard this approach for data management needs.*

While the representation issues in this context are significant, it is certainly possible, and in many cases profitable to use an RDBMS for managing scientific data. The Human Genome Project has a wide range of data requirements, from acquisition, to high level search and browse, to detailed analysis. The data is highly structured and interrelated, from the genomic data, to protein structure and sequence, to disease, and various functional data. In all, there are upwards of several hundred independent datasets in the community. While there is no standard set of tools used in this community, what is seen more often than any other choice is the use of RDBMSs. The use of relational systems in this community continues to grow. A starting point for related information is `http://www.jgi.doe.gov/JGI_home.html`. The EOSDIS project is another example of a scientific community using commercial relational technology. Information on this project can be found at `http://spsosun.gsfc.nasa.gov/New_EOSDIS.html`.

RDBMS 9 Myths:

> *"RDBMSs are fast due to excellent query optimization benefits" This is just simply not true, as the performance study below details. Query optimization speeds things up, certainly, but*

not yet close to the capabilities of the I/O subsystem. DBMSs are still overwhelmingly compute-bound.

RDBMSs are still overwhelmingly compute-bound. There are many indicators of this, from the small ratio of actual throughput versus potential throughput achieved in the performance tests ran above, to the limited impact of the cache.

In processing a query, there are many steps that take place. First, the query optimizer is run at the server. This represents part of the overhead for the entire query, and for large runs is a minimal part of the overall cost. A large part of the code path *per tuple returned to the client* involves locking, alignment, transaction management, tuple management, page management, and update, delete and insert facilities. Some relational systems require that each tuple returned to the client (in this case, our ICDA application) involves calls to the server that invoke much of the code path listed above. Other systems allow one server call to fill up large buffers and pass those back to the client. Independent of which RDBMS is used, however, due to the transaction semantics and consistency and correctness guarantees made by the DBMS, the result is that only a small fraction of the potential I/O bandwidth is used in responding to a query, even for large range and multipoint queries.

ORDBMS 1 Advantages:

ORDBMS combines, in theory, the modeling capabilities of ODBMSs with all the RDBMS advantages mentioned above.

ORDBMS offer ODBMS-like modeling capabilities by allowing the user to extend base types with user defined types, to create and query over complex objects, to specif inheritance, and to specify rules or constraints over these objects. These extensions should help remove the limitations discussed in *RDBMS #6* above. The ORDBMS then goes to great pains to build the objects into the relational model and the relational quer optimizer, in a way that allows the system to support ad-hoc queries. This overcomes what has been one of the main ODBMS limitations. Finally, a few of the ORDBMS vendors have re-worked their architectures to such an extent that near-NU performance should be possible in the future.

There are several smaller vendors of ORDBMSs on the market, and three giants: IBM DB2, Oracle 8, and Informix IUS. DB2 is using its DataJoiner technology. Oracle is providing OR capabilities through *cartridges*, which are essentially component technology for DBMSs. Informix is providing a similar approach called *data blades*. Some details of an in-depth ASCI SDM evaluation of these products can be found at `http://www.ca.sandia.gov/ASCI/sdm/`. For evaluation purposes, we have worked with the Informix IUS. The reasons are: the IUS was available; the architecture has a few special features that make the IUS extremely attractive for ICDA applications; we had access to one of the core developers of IUS functionality at Informix; and to a point the issues we explored within are common to any ORDBMS product.

The IUS allows the user to build user-defined types (UDTs), and provide methods for accessing the data in those types. The system also provides hooks for presenting an interface to those new objects in terms of relational tables. This interface includes information that helps the query optimizer make better use of the new UDTs in responding to queries. The data blade approach allows collections of UDTs and the corresponding interface to those UDTs to be grouped into a package, and provided as an add-on to the DBMS. The hope is that 3rd party vendors will get interested in building blades, and selling them to ORDBMS customers. To an extent this has already started happening, and now there are blades for spatial data, text, digital media, and various vertical industries.

One of the special features of the IUS is its underlying technology called the *Virtual Table Interface*, or the **VTI**. The VTI is the mechanism IUS uses to define the interface of the UDTs to the database. The interesting aspect of the VTI is that it allows the data being described in the UDTs to sit outside the DBMS storage manager. In practical terms, for an ICDA application, this means that the original large binary object that legac ICDA applications work on does not need to be copied or modified. Legacy applications will continue to work as before. The VTI essentially provides a way to wrap that binar object in such a way that the ORDBMS can see inside it, and provide query access to that data. This way, the query capabilities of an RDBMS are added to the ICDA applications without impacting the storage costs or access speeds for legacy data.

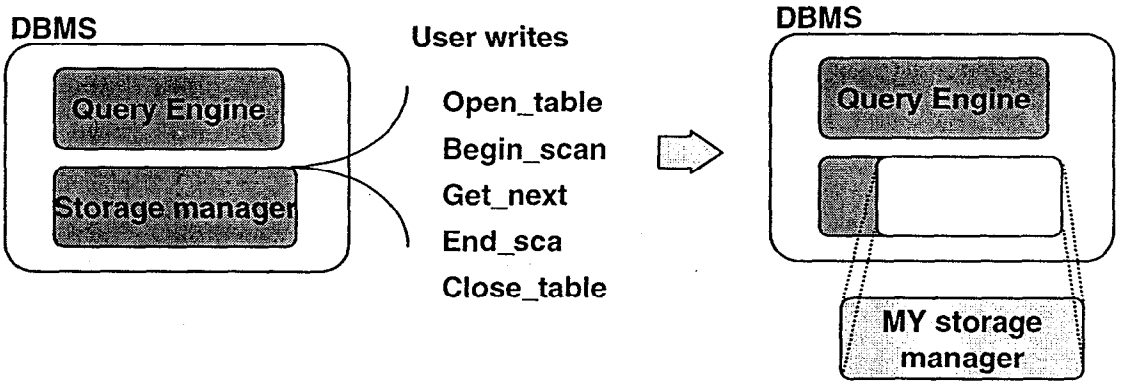


Figure 5

At a very high level, an RDBMS architecture has a query engine, and a storage manager. The VTI allows a good part of the storage manager to be defined as part of the interface to data that sits outside the realm of the DBMS.

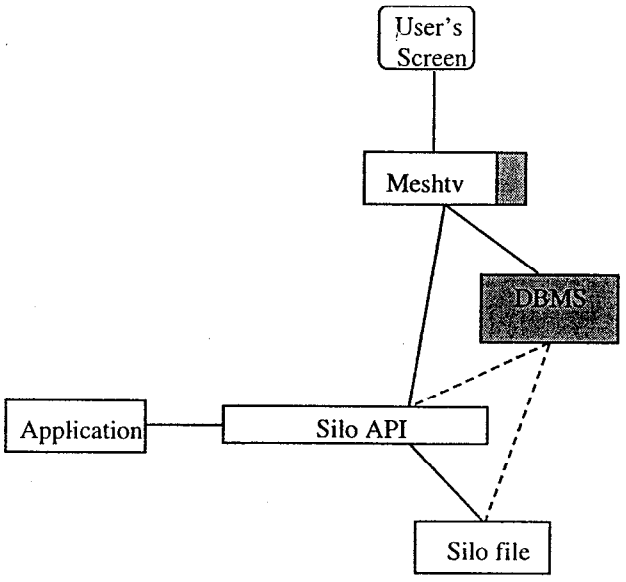


Figure 6

This figure shows an ORDBMS addition to a typical ICDA scenario. Silo is the native ICDA data format, and an API is available. Meshtv is the visualization tool. Two paths to the data are provided. If the query or visualization request can be answered by the legacy system, the Silo access path is used. If the request is for new functionality, the the ORDBMS path is used. Note that the ORDBMS can be defined in terms of the Silo API.

ORDBMS 2 Advantages:

> ***ORDBMSs have fantastic potential for speed. As tested below, the approach is within 3-15x of native Unix. However, the approach is capable of providing near-native Unix fread and fwrite performance levels.***

The IUS VTI approach (defined in *ORDBMS #1* above) has many implications. First, it means that using the VTI requires one to build the access methods to the external data that basically provides a new storage manager for that data. This provides a great deal of flexibility, but can also be a lot of work. The benefit is that much of the RDBMS code path involving locking, page management, update delete and insert facilities, and etc. will be avoided. Furthermore, if handled properly, the communication between the server and the client can be cut tremendously. For large requests, the throughput capability of this new approach should be capable of reaching near-NU performance.

Performance testing was done comparing the IUS to relational systems, and to NU. The tests were run in the same format as described above in *RDBMS #5*, with the main modification being that the data model was changed out and replaced with a more object-relational approach. A full silo object-relational test was partially constructed, and can be found at `http://www.llnl.gov/*/dbms/vti_code.html`. The testing that was done was performed on a much simpler binar dataset of our design (a set of vector variables and a corresponding schema and interface), and can be found at `http://www.llnl.gov/*/dbms/or_perf.html`. The simpler data was used since it removes several uncertainties (schema design, interface design, query design) from the testing process, and allows us to focus more on performance of relevant queries.

Summarizing, the IUS ended up showing a 3-5x performance enhancement on the range and multipoint queries that were run against it. Unfortunately, it still ends up being 3-15x slower than NU, depending on the query and the NU implementation being used. We show an example of the test results below, more can be found at the web site above.

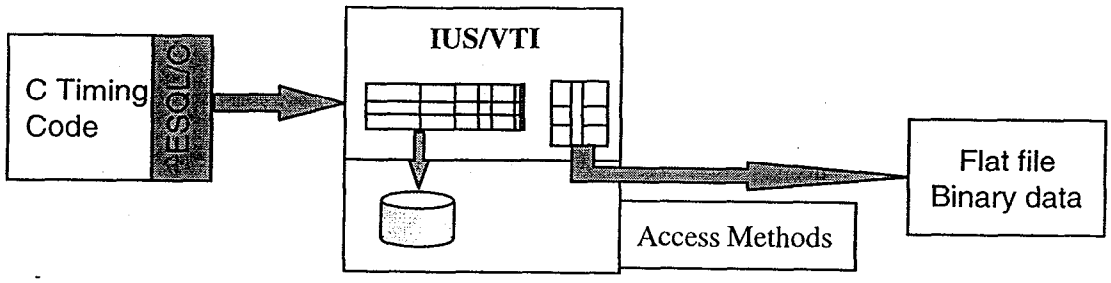


Figure 7

This figure shows the testing architecture. There is C timing code, with an ESQL interface to the ORDBMS. In the ORDBMS, the interface to the external flat file binary data is defined by specifying a set of relational tables that describe the data characteristics of interest, as well as the access methods that allow us to read the data. The tables for the external data can be mixed at will with the standard RDBMS tables.

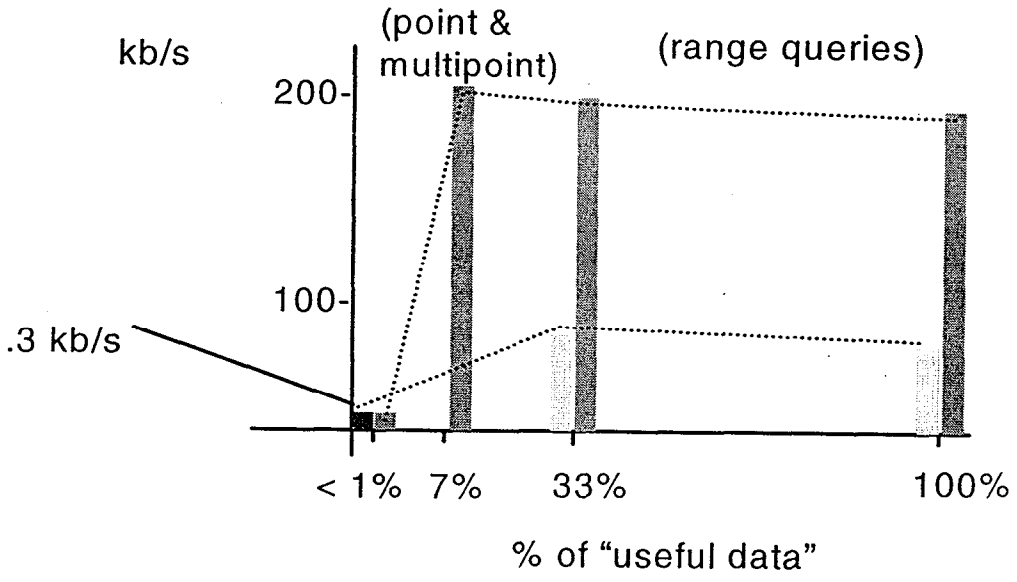


Figure 8

This figure shows a comparison of the IUS with an Informix and a Sybase RDBMS. The relational systems performed similarly. The good news is that the IUS is 3-5x faster than the relational systems. It is still 3-15x slower than NU, especially at the

larger range queries. IUS throughput at "100% useful data" is only 2% of the maximum apparent bandwidth, vs. about 45% at "7% useful data".

There are, however, several reasons for optimism here. First, since the ICDA data is stored outside the DBMS in its original format, legacy applications run at full speed. This alone significantly reduces the importance of the performance issue, since a user will be willing to tolerate slower speeds for new (and otherwise unavailable) data exploration capabilities. Second, there are several areas where the current version of the IUS fails (miserably) to live up to its potential. For example, currently the VTI allows the user-defined access methods to prefilter the data on the client side. This is good, since the server need not be called for ever tuple. It still, however, requires every qualified tuple to have an expensive interaction with the server. When a tuple is at the level of an element of an array, this is very bad news. This level of interaction is not a feature of the VTI interface, but rather the current implementation of it. Informix expects this particular issue to be resolved quickly.

ORDBMS 4 Advantages:

This approach will be part of what eventually unites the ICDA and the OLAP data management systems.

The new ASCI data model, the vector bundle interface, may be to scientific data as relational calculus and relational algebra were to business data. Relational calculus provided a common underlying mathematical basis for RDBMSs. This in turn lead to a host of 3rd party tools that, to a large degree operate on any relational data. The VB could provide this mathematical basis for ICDA data and applications. The ORDBMS would be the natural place to build the VBI into the relational model, by taking advantage of Oracle cartridges, or Informix blades. It may even turn out the data model that works for ICDA data would also be idea for OLAP applications that currently are relying on multi-dimensional database or data cube technology (both worlds need to be able to hand large arrays of numbers effectively).

ORDBMS 5 Disadvantages:

There are no tested, bullet-proof approaches yet for how to integrate the OO objects into the relational query interface. This means that any application that relies on one style of interface will be largely tied to a single vendor.

The current mode of integrating the systems is to pull the OO data into the relational side, and try to treat it as much like relational data as possible in the query engine. A relational interface is built for the OO data that wraps it, providing 1) a set of relational tables describing the data, and 2) a set of access methods for working with the data. At this abstract level, the large ORDBMS vendors look very similar. At the detail level of how this interface is actually specified, created, and used, there is little vendor agreement. Applications that go with one vendor are likely to be tied to that vendor for several years. Different ways of integrating the two approaches are bound to come up.

The current design has several weak points. First, query optimization is difficult now. The designer has the non-trivial task of collecting and providing the information the query optimizer needs in order to build the new OO data into the relational "fold". Second, it is no longer easy to do several things that are natural in an RDBMS environment. For example, there is no one line SQL command to create an index for the OO data that has been tied in.

Third, by wrapping the OO data and accessing it through the relational system, much of the potential flexibility of the system is stunted. While we have full flexibility in designing the external objects and access methods, the interface to that data must be built in terms of relational tables and external functions, which is limiting. For example, consider a large 2-D array stored outside the ORDBMS. To build this into an ORDBMS, there are several choices. A typical relational approach could be followed that has table with columns: "i", "j", and "value". This would work, and would allow full query access to the data, but does not take full advantage of the new OO capabilities. Instead, functions could be constructed that provide access to the object, like "retrieve" that takes as arguments (array_name, i, j), and returns the value. The downside of this approach is that any query that attempts to access the data in a way not preconceived and pre-supported through the functions in the interface may require new code to be written. In other words, the data is not explicitly represented (as with relational tables), and so true ad-hoc queries over the data may not be possible with "simple" SQL statements.

In conclusion, the OR connection, while interesting, raises many sticky issues due to the difference in culture, capability, and basic data model. These issues will take time to iron out in such a way that the vendors have a consistent, standard powerful approach.

ORDBMS 6 Disadvantages:

> *This is a new approach, and the instantiations of it being offered by the major vendors are not ready yet for general consumption. There is no delivery yet on the potential advantages mentioned above.*

The ORDBMS has great potential, but at this point, none of the advantages described above are actually achieved. First, the modeling capabilities of ODBMS are onl partially accounted for. The RDBMS advantages of market, external vendors, standards and maturity are currently RDBMS advantages alone, and are not shared by the ORDBMS products. Second, the ORDBMS can be made faster than the corresponding RDBMS for ICDA applications, and should eventually reach near-native unix speeds. Right now however, this goal is still a distant one, as shown above. Third, while ORDBSs may be the wave of the future, and may enable a unification of ICDA, OLAP and OLTP data management systems, neither statement will actually come to pass within the next few years.

Beyond not yet delivering on the potential advantages, the current products are complicated pieces of code, and are introducing a completely new way to interact with,

and manage data. The products are not mature, and the new OR mode of interacting with data is neither well nor widely understood.

ORDBMS 8 Disadvantages:

> *The interface to the capabilities is buggy, fragile, limited, and at times completely bewildering.*

There are three main issues that are discussed here: the unwieldy interface for utilizing the OR capabilities; the buggy, fragile and limited nature of the interface, and the overwhelming set of the data modeling options that the interface opens up. The specific examples used in this section to illustrate points concern the Informix Universal Server, as that is the system in use for the case study. However, based on the literature, and our experiences with the relational products from all the major vendors, it is safe to say that the conclusions stated above will apply equally well to any of the three large ORDBMS vendors.

The current interface for the IUS to tie external data into the relational system takes a great deal of effort on the part of the DBMS developer. To internalize a Silo file, or an other external object, several chunks of code need to be written. Examples of this can be found at `http://www.llnl.gov/*/dbms/vti_code.html`, and can be better understood from the following Figure. To build the interface to the Silo data, we need to:

1. Develop the high-level interface that ICDA users would use, which behind the sheets calls the interface defined in #2 and #3. For our case study, this would involve modifying the visualization products with some new features that allow different exploration of the mesh, supported by the ORDBMS.
2. Define the new relational tables that define (in conjunction with the new user defined functions) the interface to the Silo data. The difficulty of the design depends on the specific project. The implementation is fairly simple, in SQL.
3. Define the new functions that complete the interface to the Silo data. Both the design and the implementation difficult is driven primarily by the functions being created.
4. Define the access methods used to replace DBMS-level storage functionality. The access methods are defined at a high level, to the ORDBMS. Methods are needed for things like open_table, begin_scan, get_next, end_scan, close_table, and etc. The definition includes function names, locations of the code, and etc. Implementation is in extended SQL.
5. Write the code for the access methods above. These methods can be arbitraril complex, depending on how intelligent the memory management and disk management is, how complete the qualification system is, and how complex the external data is. The implementation must be done using a set of libraries that replace the Unix-based file and memory management routines with Informix versions of the same, since the IUS runs in a CPU extended virtual process.
6. Compile the access methods, and place them in the DBMS extension code.
7. Reconfigure the DBMS to create and handle the extent virtual processes properly.

8. Create tables in the ORDBMS according to the definitions above, and you are ready!

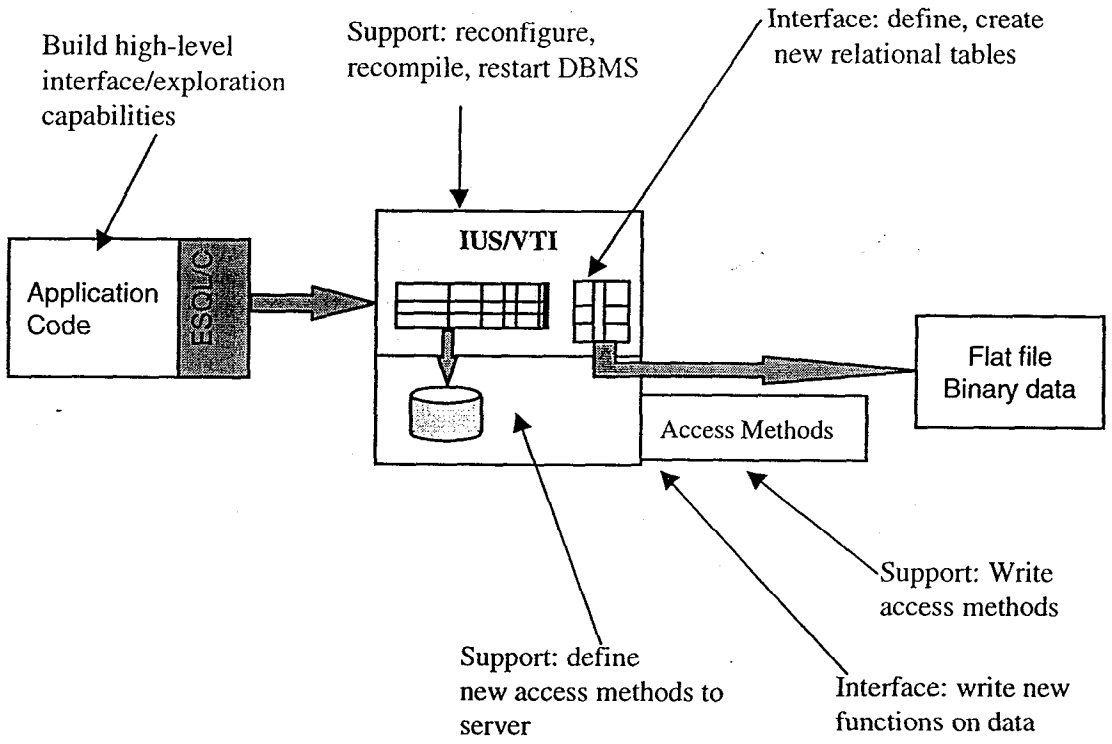


Figure 9

The second issue centers around the fragility and limitations of the interface. There were quite a few bugs and VTI design problems, some more serious than others. We were using IUS Version 9.12.

1. The VTI use of a CPU extent virtual process causes several *serious* difficulties. First, standard RDBMS mirroring facilities do not work here. Second, while POSIX operations are supported, Unix file system operations are frowned upon, and *alloc is forbidden. When writing the access methods (step 5 above), this means all interaction with external data must be written from scratch using the Informix libraries. In ICDA applications, this means that a large base of legacy code will not be available for use. For example, in writing the access methods for Silo data, the most beneficial course of action (in many ways) would be to use the Silo API in the ORDBMS access methods. This is not an option here.
2. Bugs in the code for the access methods can corrupt the database. During the course of the case study, the test database (and unfortunately, once the public database) had to be rebuilt from scratch literally hundreds of times. This makes the potential consequences of buggy user code high for ICDA applications, and should give serious pause to users.
3. There are several annoying aspects of the interface. For example, in the access methods that we write, every value returned to the server needs to be strongl typed,

then loaded into a "pdatum" structure and sent to the server. This is slow, due to the server communication requirements. It is also annoying since we will not know ahead of time whether the particular mesh variable of the particular mesh we are reading is int, float, double, or etc. This means a lot of ugly code in the access methods to handle type issues.

4. There were a host of small bugs in the IUS code, including a broken qualification evaluation routine, incorrect specifications for the various machine architectures (that lead to database-corrupting crashes), and faulty type evaluation code.

The third issue is that the extent of new options that are opened up to the users is bewildering. For the IUS, data can be stored with relational tables consisting of a variet of standard basic types, as well as money, and date types. Then there are row types, blobs, smart blobs, opaque types, distinct types, collections, sets, data blade module data types, and user defined functions and casts. Each of these options has a unique set of capabilities attached to them, such as query flexibility, bulk loading capability, different access methods and indexing options. They have varying degrees of self description, ease of use, and potential performance. Starting from knowledge of the RDBMS, or ODBMS worlds helps, but the learning curve is steep. Beyond that, the question of how to create a good *object-relational* design is largely unexplored territory. This is one of the most significant, and yet subtle dangers for the ORDBMS community. Powerful, but overl complex and hard-to-use technology is a niche commodity at best.

ORDBMS 10 Myths:

> *"Saves time" At this point, utilizing the OR aspect of the system to provide access to ICDA data is so difficult, that the time might be better spent being amortized over many smaller projects focused at extending legacy ICDA systems to allow ad-hoc queries, indexes, and etc.*

The ORDBMS interfaces currently require a lot of effort to make appropriate use of. If an ICDA application simply wants to provide ad-hoc query access to the data, *and nothing more*, a better choice at this point in time might be to implement such a system from scratch without using a DBMS. This would require the design of simple quer language, and a parser that implements it (this can be done in a day or two using Lex/Yacc, see example at `http://www.llnl.gov/*/dbms/parser.html`), along with code to read the ICDA data format in question (i.e. Silo) to serve the data. This code is similar to that needed to supply the access methods for get_next for the ORDBMS code.

ORDBMS 11 Myths:

> *"More flexible" In building the oo object into the relational system, several restrictions are placed on the object – views if you like. The views limit the flexibility that might otherwise exist in the object. Furthermore, the access method that is created to support the external object will only be effective under certain rigid conditions.*

This "myth" has a more uncertain status than others. We have argued in this text that the number of options open to the developer is so large, that it is nearly overwhelming. But, as described in *ORDBMS #8,* these options each carries its own set of limitations, and the

end result is that the current O-R integration approach introduces limits on the overall data model. Given a good design, the ORDBMS interface should provide very flexible access to the underlying ICDA data, much more so than the pure functional interfaces that are standard in the community. This is because query access to the raw data is possible in the ORDBMS world, whereas in the standard ICDA framework, what is presented is a pre-conceived set of functions that provide some defined subset of functionality over that data. If the ORDBMS design instead provides an interface based primarily on functional interfaces to the external data, then not only is there little advantage in terms of user flexibility, but also the cost of modifying or extending the interface is likely to be much higher on the ORDBMS side. The point here is that ORDBMS is not a magic bullet. Like any system, the new capabilities it brings to the relational DBMS world must be well used before they add value.

5 Discussion

One thing that became very clear during the evaluation is that there are three prerequisites that must be met simultaneously in ICDA applications. Unfortunately, every approach to data management seems only to be able to support two of the requirements concurrently. The prerequisites are: (1) small storage costs, (2) standards-based ad-hoc query support, and (3) reasonable throughput (Mb/s) performance. Most combinations of two actuall seem to be supported. For example, blobs do not consume much overhead storage, and have reasonable throughput performance, but do not provide good SQL access into the file. Alternatively, by keeping a copy of the data (thereby doubling or tripling storage requirements) in the original flat-file format outside the DBMS, legacy applications such as visualization will not see a performance degradation, and will get the ad-hoc quer capability on the data stored in the DBMS to boot. This observation holds for the current legacy systems as well. They support low storage costs with great performance, but do not support standardized ad-hoc queries against the data.

To support ICDA applications like those found in ASCI, what is needed is a large, stable vendor that provides fully a operational query engine on top of a data server with flat-file like performance. This (hypothetical) system would probably have features like:

1) Large, multi-dimensional array data stored *outside* the DBMS storage manager. This way, external applications can use the data without taking a performance hit, and without the large storage overheads.

2) Mechanisms to plug this data into the query engine, in a way that queries can combine data stored in the DBMS with data external to it. If the standard SQ interface is to be used, this might require being able to create relational views of the external data, and then populating the views on the fly when a query hits the external data. Population would allow passing large collections of data from the external access methods, rather than passing 1 tuple at a time.

3) Avoidance of as much of the DBMS code path as possible. To protect the performance gains, have the hooks in #2 be inserted so as to avoid as many of the

following as possible: locking, alignment, transaction management, tuple management, page management, insert and delete facilities. These features are much less interesting when there are few users on WORM data that is stored *outside* of the DBMS.

ORDBMS vendors are beginning to close in on these targets, and may be viable options for supporting ICDA applications within 5-7 years. Technologies and trends to keep an eye on in the business community include:

1. Clearly, ORDBMS technology should be closely watched, as currently this is closest to fulfilling ICDA requirements.
2. The response of the ODBMS vendors to the new ORDBMS competition. They might be pushed into a new design cycle that leapfrogs current ORDBMS status.
3. The datacube and multi-dimensional database approaches to OLAP, data mining and other business intelligence applications. Datacube and MDDB approaches work on large multi-dimensional data that can be numeric or text. They may grow into providing much of the functionality needed for ICDA, and may overtake much of the related market for data mining support in the business community. If they do, tools will be available that will enable easy communication between DBMSs and these tools, since business data will be their main driving force.
4. The progress of underlying scientific data models and formats to a common standard, such as something based on the vector bundle interface. A common underlying data model alone might be enough to be able to easily make use of commercial scientific visualization packages like EnSight, and others that share that model. While this won't realize all the potential benefits of being able to use commercial business DBMS technology, it would be a good start.

Acknowledgments

This work has benefited tremendously through discussions with Mark Miller.

Bibliography

[BS95] P. Brown and M. Stonebraker. BigSur: A System for the Management of Earth Science Data. In *Proceedings of the 21st International Conference on VLDB,* Zurich, Switzerland, 1995.

[Cat96] R. Cattell (editor). *The Object Database Standard: ODMG-93, Release 1.2,* Morgan Kaufmann, San Francisco, 1996.

[CGM98] T. Critchlow, M. Ganesh, R. Musick, K. Fidelis, and T. Slezak. DataFoundry: Warehousing Techniques for Dynamic Environments. Submitted to *International Symposium on Molecular Bioinformatics,* 1998.

[CL97] A. Chaudhri, and M. Loomis. "Object Databases in Practice", Prentice Hall, New Jersey, 1997.

[GC97] S. Goil, and A. Choudhary. High Performance OLAP and Data Mining on Parallel Computers. *Data Mining and Knowledge Discovery Journal* , 1:4, 1997.

[Goo95] N. Goodman. Tutorial: Research Problems in Genome Databases. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* San Jose, CA, 1995.

[Gra93] J. Gray, editor. *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, San Francisco, 1993.

[HDF98] HDF home page can be found at: http://hdf.ncsa.uiuc.edu/

[Kin93] D. Kingsbur . *Report of the International Workshop on Genome Informatics.* http://www.ornl.gov/TechResources/Human_Genome/publicate/miscpubs/bioinfo/inf_rep2.html# , 1993.

[MM97] D. Malone and E. May. Critical Database Technologies for High Energ Physics. In *Proceedings of the 23rd International Conference on VLDB* , Athens, Greece, 1997.

[NAS91] NASA, *EOS Reference Manual* , NASA Goddard Space Flight Center, Greenbelt, MD, 1991.

[OLA98] OLAP Council Benchmarks found at http://www.olapcouncil.org/research/bmarkly.htm

[RD96] D. Rotum and J. Zhao. Extendible Arrays for Statistical Databases and OLAP Applications. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management,* Stockholm, Sweden, 1996.

[RDE93] R. Rew, G. Davis and S. Emmerson. *NetCDF User's Guide, An Interface for Data Access, Version 2.3.* Unidata Program Center, Boulder, CO 1993.

[Sha92] D. Shasha. "Database Tuning, a Principled Approach". Prentice Hall, NJ, 1992.

[SM96] M. Stonebraker with D. Moore. "Object-Relational DBMSs: the Next Great Wave", Morgan Kaufmann, San Francisco, 1996.

[TD96] R. Tanler and K. Drost. Multidimensional Analysis of Warehoused Data. "Data Warehousing: Strategies, technologies and techniques", Rob Mattison. McGraw-Hill, New York, 1996.

[TPC98] Transaction Processing Council benchmarks found at http://www.tpc.org.

[WD92] S. White and D. Dewitt. A Performance Study of Alternative Object Faulting and Pointer Swizzlin . In *Proceedings of the 18th International Conference on VLDB,* 1992.

[ZCF97] C. Zaniolo, S. Ceri, R. Snodgrass, V. Subrahmanian, R. Zicari. Advanced Database Systems. Morgan Kauffman, 1997.